The Thesis Committee for Dominick Anthony Mulder
certifies that this is the approved version of the following Thesis:

Development of Algorithms for Service Robots in Domestic Environments

APPROVED BY

SUPERVISING COMMITTEE:

Luis Sentis, Supervisor

Farshid Alambeigi

Development of Algorithms for Service Robots in Domestic Environments

by

Dominick Anthony Mulder

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2020

Dedicated to my wonderful family.

Acknowledgments

I would like to thank my advisor, Luis Sentis, for all of his guidance throughout my time working in his Human Centered Robotics Laboratory. It is shocking to think about the amount of knowledge, skills, and experience I have gained since joining his lab. Beginning with his recommendation for me to join to Austin Villa RoboCup team, he has consistently offered me support in my academic pursuits and has provided essential advice towards career development.

I would like to thank Farshid Alambeigi, whose lessons in his Algorithms for Sensor-Based Robotics course have taught me essential skills which will be instrumental in my future endeavors. The knowledge which I have obtained through this course has provided me with an amplified excitement towards potential projects relating to robotic systems.

I would like to thank Justin Hart and Peter Stone for their leadership during the RoboCup@Home project. This project was essentially my introduction to state-of-the-art tools used in robotic development, and their support has opened me up to the world of robotic systems and AI.

I would like to thank my peers in the Human Centered Robotics lab and from the Austin Villa RoboCup team. Thank you to Rachel Schlossman for guiding me as I joined the lab with limited experience in robotics development.

Thank you to Gilberto Martinez, Ryan Gupta, Haresh Karnan, Rishi Shah, Yuqian Jiang, and Marika Murphy for collaborating with me in preparation for the RoboCup@Home competition. Thank you to Nicolas Brissonneau and Steven Jorgensen for discussing ideas with me throughout my time developing Machine Learning algorithms. Thank you to Minkyu Kim, Henry Cappel, Mihir Vedantam, Huang Huang, Jee-eun Lee, Binghan He, Junhyeok Ahn, Jaemin Lee, and Seung-Hyeon Bang for being helpful and friendly lab-mates.

I would like to thank Michael Haberman, Raul Longoria, and Randy Williams, with whom I have worked as a Teaching Assistant. They have all been a pleasure to work with and I am grateful for the skills they have helped me to develop, not only in improving my fundamental understanding of dynamic systems and controls, but also with my communication skills which have improved while regularly working with students.

I would like to thank the rest of the University of Texas at Austin faculty for their instrumental roles in my academic and personal development. I am grateful for the valuable content of their coursework and the advice they have consistently offered to me.

I would like to thank the friends I have made during my time living in Austin, both inside and outside of the University of Texas engineering community. Their camaraderie and support have been essential in my growth as a student and as a person, and for encouraging me to work towards my goals.

Finally, I would like to thank my parents, Jennifer Chirolis and Alfredo

Mulder for the unwavering support and love throughout my academic career. It cannot be emphasized enough how much they mean to me and how grateful I am to have them in my life. Thank you as well to the rest of my loving family for their support.

Development of Algorithms for Service Robots in Domestic Environments

Dominick Anthony Mulder, M.S.E.
The University of Texas at Austin, 2020

Supervisor: Luis Sentis

This thesis focuses on developing software for mobile robots which enables these platforms to perform household tasks. A set of tasks was proposed by the RoboCup Federation for the 2019 RoboCup@Home competition. The primary goal of this competition is to develop a unified robotic system with the capability to assist humans in their daily lives. Efforts toward this competition were made through collaboration with the Austin Villa team at The University of Texas at Austin. Toyota's Human Support Robot served as the robotic platform for the competition. This thesis primarily focuses on a task in which the robot autonomously collects trash bags and transports them to a designated location.

Cooking is another household task addressed in this thesis. In particular, this work examines methods of enabling a robotic system to understand the cooking process and use this understanding to make informed decisions.

The case study that was analyzed involves using a Convolutional Neural Network to process image data and estimate when a pancake has been sufficiently cooked.

The work in this thesis contributes toward development of tools for mobile service robots in domestic environments. These tools are also implemented and shown to be effective in enabling robots to improve quality of life by assisting in household chores.

Table of Contents

List of Figures

Chapter 1

Introduction

Advanced technology has become a regular part of life for many people. Smartphones, Virtual Assistants, and Smart Appliances have found their way into households around the world. This thesis examines the development of tools which would work towards allowing mobile service robots to similarly assist humans around the house and in their daily lives.

RoboCup@Home is an annual competition which encourages the advancement of such tools. In this competition, the RoboCup Federation proposes a set of tasks which are intended to assist humans. These tasks are especially motivated by the desire to help those with illnesses or disabilities. The overall goals of this league include development towards Computer Vision, Object Manipulation, Safe Navigation, and Task Planning, as well as Human-Robot Interaction capabilities such as Speech Recognition and Person Detection [15]. Chapter 2 discusses the work done towards the RoboCup@Home competition, with emphasis on the task of Taking out the Garbage.

While not part of the RoboCup@Home competition, cooking is another household task in which a service robot may provide assistance to humans. For an autonomous robot to be reliable during a cooking task, it is necessary

for it to have an understanding of the cooking process which allows it to make effective decisions. Chapter 3 examines the case study of using a Neural Network to use visual data to estimate the state of a pancake as it is being cooked, in order for the robot to be able to identify the proper time to flip the pancake. Without an ability to dynamically understand its environment, a robot would need to depend on an event occurring exactly the same way every time. In this case, a simple tool such as a fixed timer would be sufficient. This thesis aims to develop methods which result in more robust performance by enabling the robot to react to external variables.

Many robots are designed with a highly specific application in mind, such that they perform exceptionally well for that purpose. The primary goal of this thesis is to make progress toward a unified robotic system with a diverse set of functionalities which can provide support for humans, particularly those with limited capability to support themselves. Chapter 2 addresses the use of Toyota's Human Support Robot (HSR) as a general-purpose service robot, by combining Task Planning, Manipulation, Perception, and Navigation to perform household maintenance. Chapter 3 focuses on developing Computer Vision and Deep Learning algorithms which can readily be applied to mobile robots such as the HSR to enable it to estimate the state of a food item as it is cooking.

Chapter 2

Software Development for a General-Purpose Service Robot

2.1 Summary

RoboCup is a worldwide competition with the primary goal of advancing the capabilities of robotics and Artificial Intelligence[1]. RoboCup is largely popular for its Soccer tournament, in which participants develop strategies for their team of autonomous robots to compete in Soccer matches. Other branches of RoboCup focus on rescue and industry tasks.

This chapter focuses on work done towards the 2019 RoboCup@Home competition, in which teams work to enable a mobile robotic platform to perform tasks in domestic environments such as restaurants, malls, or homes. Specifically, this software was developed for the Domestic Standard Platform League, in which each participating team uses the Human Support Robot (HSR) from the Toyota Motor Corporation (TMC) as their robotic platform. The specifications and capabilities of the HSR are discussed in Section 2.3.

The RoboCup@Home competition primarily takes place in an arena which consists of a realistic apartment setting, containing common furniture

[1] `https://www.robocup.org/`

such as a couch and cupboard, as well as typical household objects such as bowls and bags. The 2019 competition tasks were categorized as either a Housekeeper or a Party Host task. Housekeeper tasks focused mainly on cleaning and maintenance, while Party Host tasks involved more direct interaction with humans [15]. This chapter takes a detailed look at the Housekeeper task of Taking out the Garbage.

2.2 Contributions

The software used for this competition was developed largely through the collaboration of the Austin Villa team from the Department of Computer Science at The University of Texas at Austin. This team was led by faculty members Justin Hart and Peter Stone, and student members included Rishi Shah, Yuqian Jiang, Haresh Karnan, Gilberto Martinez, Ryan Gupta, Rachel Schlossman, and Marika Murphy. This team developed a framework and tools which were instrumental in approaching the goals of this competition as well as advancing the capabilities of interactive robots. The contributions of the Austin Villa team are discussed further in Section 2.4.

My specific contributions to the project revolved largely around the Take Out the Garbage task, as detailed in Section 2.5, particularly towards ensuring the Navigation and Manipulation components were safe and worked consistently. I worked alongside Haresh Karnan for this task, who assisted me in the development of the overhand grasp method discussed in Section 2.5.2. Haresh Karnan, Rachel Schlossman, and the rest of the Austin Villa team also

assisted in applying the tools from ROS and the tools developed by the team.

2.3 Human Support Robot

The Toyota HSR was designed as a platform for addressing the needs of the elderly and disabled. It is not intended to be a substitute for real human interaction, but it is capable of improving quality of life by assisting humans in tasks that may otherwise be difficult or infeasible. The HSR is shown in Figure 2.1, along with its primary features and hardware [5].

Figure 2.1: Toyota's Human Support Robot (HSR) with annotated hardware and features.

2.3.1 HSR Hardware

The HSR is equipped with an omni-directional moving base designed for indoor use at a safe speed of $0.22m/s$. The torso of the robot acts as a prismatic joint which is capable of lifting and lowering its arm and head. The arm has 4 degrees of freedom, with 2 rotational joints for flexing and rolling the main arm link, and 2 additional joints for flexing and rolling the attached wrist. The end effector can provide a maximum gripping force of $40N$ and can hold objects of widths up to $130mm$. A vacuum pad is attached to the gripper which provides up to $5N$ of suction force, which can be used for picking up thin and light objects such as paper. The HSR has 2 additional rotational joints for panning and rotating its head [7].

Within the body of the robot is a 4th Gen Intel Core i7 CPU and a NVIDIA Jetson TK1 embedded GPU board. The torso also contains a Inertial Measurement Unit which aids in tracking the position and orientation of the HSR. A Force-Torque Sensor is included with the gripper which detects forces acting on the wrist [7].

Data used for perception is primarily obtained through the RGB-D Camera located above the HSR's head display. Visual inputs can also be obtained from the head's Stereo Camera and Wide-Angle Camera. An additional camera is located on the end effector, allowing a clear view of objects which are to be grasped. Speech inputs are accepted by the Microphone Array near the RGB-D Camera [7].

Attached to the moving base is a LIDAR sensor used to dynamically detect obstacles as the HSR navigates around the environment. In the event that the HSR does collide with an obstacle, a bumper sensor can detect this collision. For safety purposes, the robot is automatically stopped upon activation of the bumper sensor. There is also an emergency stop button which disables the robot immediately upon being pressed [7].

2.3.2 HSR Software

Many basic functions of the HSR can be accessed through the Python interface provided by TMC. The HSR has a neutral configuration which can be reached with a single function. Individual joint configurations for the arm and head can also be specified in this interface. The gripper can be set to a desired position, or the amount of applied gripper force can be specified. Built-in functions allow the end effector to follow a specified line or arc, which is useful for implementing simple motion plans. The omni-directional base can be controlled through this interface, by specifying a desired position and orientation either relative to the robot or relative to the world reference frame. Text-to-speech synthesis enables to robot to say a given phrase [6].

Many of the more complex tasks are performed by interfacing with the Robot Operating System (ROS). ROS provides a collection of tools which are applicable for a wide variety of robotic applications [22]. Nodes are a feature of ROS which are able to publish or subscribe to Topics to communicate with other Nodes. An example of this is is publishing a desired trajectory to the

HSR arm controller, in which a list of joint positions, velocities, and times can be specified [6].

Another tool within ROS is the use of Services and Clients, where the Client can make a request to the Server. A useful library for this purpose is **actionlib**[2]. A benefit of **actionlib** is the ability to receive feedback during the execution of a service. This is often relevant for services which may require cancellation before the goal is reached, such as Navigation.

Through **actionlib**, the HSR is capable of moving autonomously to a desired pose while avoiding obstacles detected by the LIDAR, provided the pose is reachable. Navigation is more reliable with a known map of the area, which can be obtained through Simultaneous Localization and Mapping (SLAM) [12]. This technology is used to resolve the problem of a robot creating a map of an unknown environment while also recognizing its own position within this map.

The **rviz**[3] package is used to visualize the robot and dynamic obstacles within the map. A variety of other visualizations may be seen in **rviz**, such as image and Point Cloud data received from the cameras. Within the **rviz** GUI, there is a tool to quickly set a 2D Nav Goal. Upon initialization, the robot will predict itself to be at the origin pose within the map. Prior to setting any Nav Goals, it is critical to manually localize the robot using the 2D Pose Estimate tool. An example of the **rviz** interface is shown in Figure 2.2.

[2] `http://wiki.ros.org/actionlib`
[3] `http://wiki.ros.org/rviz`

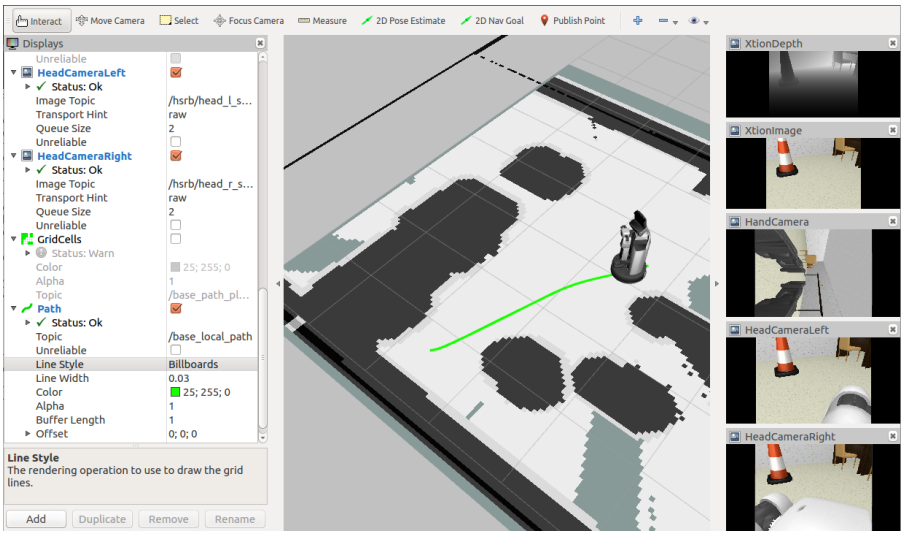

Figure 2.2: The **rviz** interface during a Navigation action. The green line indicates the path planned through the Navigation stack. The black lines indicate known obstacles from the map. The dark gray pixels show obstacles detected by the LIDAR. The 2D Pose Estimate and 2D Nav Goal tools are accessible at the top of the interface. On the right side there are several images corresponding to data received from the cameras on the HSR. On the left side there are options for customizing the **rviz** display.

2.4 Austin Villa Codebase

This section addresses the software developed through a collaborative effort from the Austin Villa team. The Department of Computer Science at The University of Texas also hosts the Building-Wide Intelligence (BWI) Project[4], which addresses many of the same goals as RoboCup@Home. For this reason, relevant code from BWI was also implemented for the competition.

[4] `https://github.com/utexas-bwi`

This section is not intended to be a comprehensive survey of all developments made by the Austin Villa team, as it focuses primarily on the features that I have worked with.

2.4.1 Architecture

In 2018, the Austin Villa team proposed a Layered Architecture for Autonomous Interactive Robots (LAAIR) which was first implemented for use in that year's RoboCup@Home competition. The top level of LAAIR is a reactive control layer which handles the overall system behavior by delegating tasks to the deliberative control layer or by calling skills. Within the deliberative control layer, the robot is able to assess its environment and call skills as deemed necessary. Skills refer to behaviors which directly interact with the environment, such as manipulating objects. Each layer is able to provide feedback to layers above it. An overview of a LAAIR system is shown in Figure 2.3 [11].

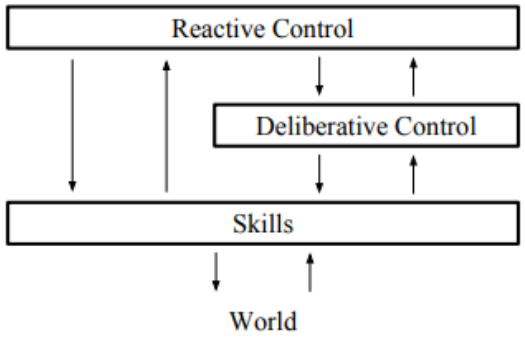

Figure 2.3: Overview of a LAAIR system.

The reactive layer is implemented through the **SMACH** library[5], which is used to generate hierarchical finite state machines. This allows a task to be decomposed into a collection of sub-tasks. In many cases, these sub-tasks may be directly completed through a sequence of skills. For more complex goals, the task planner and plan executor in the deliberative layer will be used to call and monitor the skills. A typical state has a 'succeeded' and 'aborted' transition behavior which is executed based on the feedback from the skills layer. It is also possible for an interaction with the environment to preempt execution of a state [11].

A central knowledge base is implemented to store relevant information about the environment which may be accessed by each control layer [11]. An example of useful knowledge is a collection of robot poses within the arena map. These poses are used as 2D Nav Goals and allow the Navigation stack to readily move the robot to key locations. The knowledge base can also store known attributes, such as the height of a counter-top.

2.4.2 Perception and Manipulation Stacks

Object Detection is executed through You Only Look Once (YOLO) software, which is used to detect and classify objects in real time [20]. Image and Depth data is continuously collected and processed through the Perception stack. Through YOLO, a 2D bounding box is generated around a detected object and the depth data can be used to extrapolate a 3D point cloud of the

[5] `http://wiki.ros.org/smach`

object [24].

The Austin Villa team has developed useful tools for quickly training specific objects to be recognized by YOLO. Raw images are manually captured such that the object is displayed from multiple unique angles. These images are then annotated and the background is removed with a grabcut tool. Once this process has been completed for all desired images, augmented training data can be generated through Austin Villa's YOLO pipeline. This software randomly scales and positions the annotated images and imposes them upon a collection of background images. This process results in quick artificial generation of large amounts of training data.

For Object Manipulation, the end effector is projected onto different positions of the 3D point clouds generated through YOLO to determine plausible grasp poses. From there, the MoveIt framework is used to generate the necessary motion plan [23]. While any of the selected poses will allow the target object to be grasped, the motion plan to reach this pose may be infeasible. For this reason, a layer of nodes is used in which each node attempts to generate a motion plan for a specific pose. If one pose is found to be inaccessible, that node will crash while the other nodes continue to search for a plan. The first completed plan is then chosen to be executed [24].

To address potential deviations due to imprecise odometry, the goal pose is slightly offset from the actual desired pose. This allows the object to be detected through the Hand Camera such that small changes in position can be made to align the gripper with the target object.

2.5 Take Out the Garbage

As per the RoboCup@Home 2019 Rulebook [15], the Take Out the Garbage task is a Housekeeper task in which the HSR is expected to collect two trash bags and move them to a designated collection area. The bags are each initially located in a bin, and are already tied prior to the test. The contents of the bags are light enough for the HSR to handle with ease. As a bonus challenge, lids were placed on the bins which must be removed by the robot during the test.

In the competition, the full task must be completed within 5 minutes. The HSR begins outside the entrance of the arena, and the timer begins once the entrance door is opened. To run experiments, a mock apartment was set up to emulate a typical arena which may be found at the actual competition. The map generated at the competition is shown in Figure 2.4, along with the locations of the trash cans and collection zone.

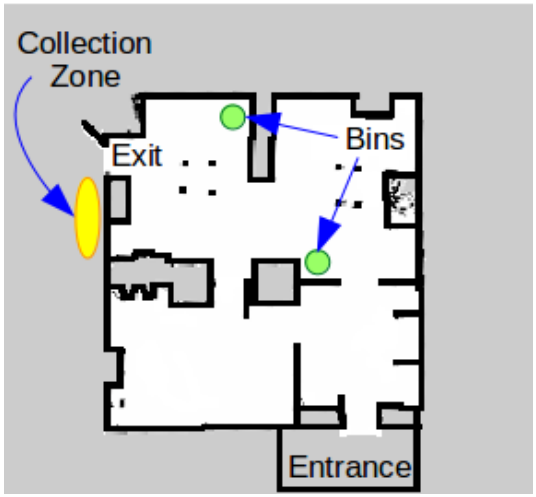

Figure 2.4: RoboCup@Home arena map with important features annotated.

2.5.1 Navigation

The map of the arena is obtained prior to the day of the competition, along with the locations of the bins and collection zone. This permits robot poses to be stored in the knowledge base and used as Nav Goals throughout the competition.

The first major sub-task is getting the robot to recognize when the entrance door is opened so that it may enter the arena. This is accomplished using the built-in obstacle detection and avoidance software referenced in Section 2.3.2. The HSR is given a Nav Goal to a pose just inside the arena. While the entrance is closed, the LIDAR detects the door as an obstacle blocking the path to this waypoint, which prevents the HSR from moving during this time. Once the door is opened, the HSR recognizes through the LIDAR that this obstacle is no longer present, and is immediately able to plan a path through the doorway. Upon initialization of the robot, text-to-speech synthesis is used to get the HSR to say, "I am ready to take out the trash." This offers clear confirmation that the robot is active while it is stationary and waiting for the entrance to open.

Once inside the arena, the HSR must move autonomously between the bins and the collection zone. Navigating to the poses stored in the knowledge base results in the HSR directly facing the bin or the collection zone, where the robot may continue with the task as needed.

One issue that presented itself during experiments was that the autonomous movement struggled when navigating near sharp corners. The planner aimed to optimize the path, meaning the HSR would travel very close to these corners if doing so would minimize the distance travelled. However, this occasionally conflicted with the obstacle avoidance software. An obstacle radius was implemented to prevent the robot from maneuvering within a specified distance from a detected obstacle. The arena walls are detected as obstacles, and the HSR had a chance of inadvertently entering this restricted zone when rounding sharp corners. Upon detecting that it is too close to an obstacle, the HSR will stop moving to ensure that it avoids collision.

While this is clearly not a desired behavior, it is unsafe to simply abandon the strict obstacle avoidance measures. Instead, it was decided that it is worthwhile to take a slightly longer path to the Nav Goal to allow the HSR to stay further away from obstacles. This was accomplished by adding additional poses to the knowledge base to serve as waypoints during Navigation. These waypoints were placed a safe distance from any known obstacles on the map such as walls and furniture. Paths can then be defined using these waypoints as intermediate Nav Goals such that the HSR stays away from sharp corners.

Another complication occurred when navigating while carrying the trash bags. It was possible for the HSR to grab the bags in such a way that the bottom of the bag covered the LIDAR when the robot moved to its neutral configuration. This not only prevented detection of actual obstacles, but also caused the HSR to permanently detect the bag as an unavoidable

obstacle. This issue was solved by slightly lifting the torso of the HSR relative to its neutral position. Figure 2.5 shows the HSR carrying one of the trash bags while navigating to the collection zone.

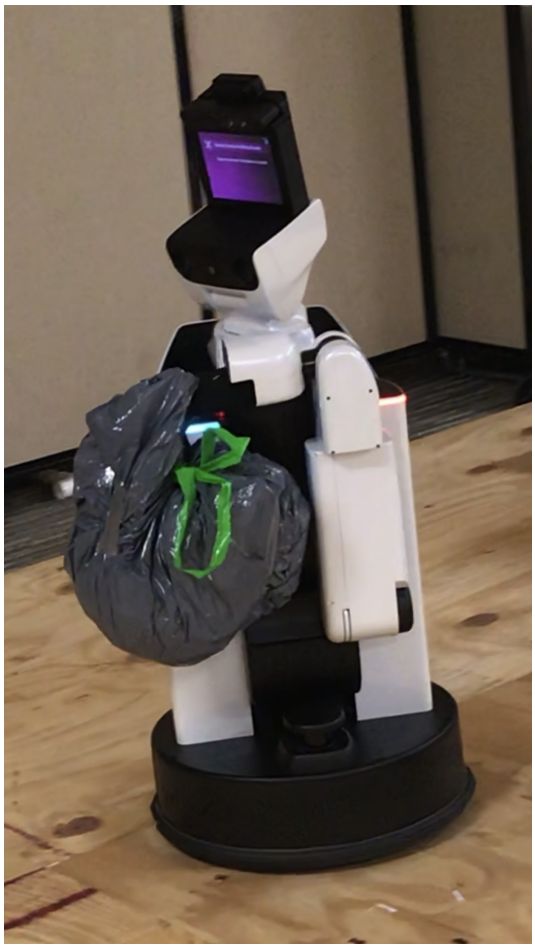

Figure 2.5: The HSR carrying a trash bag while navigating to the collection zone.

2.5.2 Perception and Manipulation

In order to manipulate the trash bags and lids, it is critical that the HSR is able to identify and locate these objects in real time. While the general locations of the bins are known in advance, they are not precise enough for Object Manipulation purposes. Once the Nav Goal corresponding to one of the bins is reached, the Perception stack takes over.

Using the process described in Section 2.4.2, images of the trash bag, bin, and lid were collected for training with YOLO. When the HSR is close enough to the bin, it is then able to detect it and a 2D bounding box is generated. An example of an image used for training is shown in Figure 2.6 [24].

Figure 2.6: A training image of a trash bag inside a bin collected in preparation for the competition.

In order to take advantage of the nature of this task, the manipulation strategies used here differ from the general Manipulation stack discussed in Section 2.4.2. For this task, it is known that the lid and bags will always be grasped from above, and the height of these objects are always known and constant between trials. While 3D Point Cloud data may be used to find a grasp pose and MoveIt can be used to execute a trajectory to reach this pose, performance was found to be more reliable when using a customized overhand grasp approach.

When using this method, image data is collected through the Hand Camera. While this does not provide depth data like the RGB-D Camera, 2D data is sufficient since the grasp heights are known in advance. Once the HSR is facing the bin, it reaches out its arm and flexes its wrist to point the Hand Camera directly downward. At this point, the bin will be detected and a 2D bounding box is generated around it. A ROS Subscriber is used to listen to the data gathered through the Object Detection node.

The objective is to align the detected bounding box with the center of the Hand Camera image. The error between the center of the image and center of the bounding box is measured in pixels. This error is used in a proportional controller to publish a velocity command to the robot which moves it toward the desired position. Once the error is small enough, the velocity command is set to zero.

The prismatic torso joint is particularly useful here, as movement along this joint causes the end effector to travel directly downward and toward the

target object. Since the hand camera is offset from the gripper, the arm flexion angle is also slightly increased during motion to account for this distance. When reaching for the bags, the arm can simply be lowered to the desired position and a force command is published to the gripper to grasp the bag. The HSR can be seen collecting a trash bag in Figure 2.7.

Figure 2.7: The HSR pulling a trash bag out of a bin after grasping it.

When grabbing the lid, the orientation of the handle must also be considered. There was difficulty in getting the lid handle to be detected through YOLO, so a different approach was taken. In the first attempt at grasping the lid, the wrist is positioned under the assumption that the handle is orientated perpendicularly to the robot, as this was considered the most likely

orientation. Once the arm was lowered to the correct position, a light force command was published to the gripper while a ROS Subscriber was used to track the joint state of the hand. Since the lid handle was very thin, a successful grasp would mean that the gripper is able to completely close. If the joint states reflected that the gripper was unable to close, it is evident that the assumed grasp orientation was incorrect. In this event, the gripper is opened and rotated to attempt a different orientation.

Once the grasp is successful, a stronger grip force is applied and the torso joint is raised slightly to remove the lid from the bin. From here, the HSR rotates toward an open area where it can set the lid on the floor. The robot then rotates back toward the bin and prepares to collect the bag. The HSR grasping and releasing the lid is shown in Figure 2.8.

Figure 2.8: On the left, the HSR attempts an incorrect grasp orientation for the lid. In the middle, the HSR correctly grasps the lid handle. On the right, the HSR sets the lid on the floor.

While this overhand grasping algorithm was developed specifically for this task, it has been shown to be useful for a variety of objects which can be grasped from above. For certain objects, such as empty bowls, the overhand grasp pose must correspond to the edge of the object rather than the center. This can be accomplished by defining the position error as the distance between a specified bounding box edge and the center of the image. In cases where the grasp height is not known in advance, the Force-Torque Sensor in the wrist may be used to detect when the gripper has contacted the target object.

The final manipulation sub-task is to drop off the trash bag in the collection zone. This is a fairly straightforward objective and does not require precise Object Manipulation as before. Once the HSR has reached the Nav Goal corresponding to the collection zone, it may simply reach out its arm and open the gripper to release the bag. The only complication that was encountered was that the bag handle occasionally got stuck on the gripper. This was resolved by implementing a bidirectional wrist rolling movement to shake the handles loose. Figure 2.9 shows this task being completed at the RoboCup@Home competition.

Figure 2.9: The second trash bag being placed in the collection zone by the HSR at the RoboCup@Home competition, indicating completion of the Take Out the Garbage task.

2.5.3 State Machine

The Take Out the Garbage task was divided into the aforementioned sub-tasks through a **SMACH** state machine as described in Section 2.4.1.

The first state initializes parameters to be used during the test and gets the robot to announce that it is ready for the entrance door to be opened. This transitions into the state which waits for the entrance door to open and then navigates the HSR into the arena. Once this state has succeeded, an internal timer is started which is used to track how much more time is allowed before the 5-minute limit is reached.

The following states move the HSR to the specified waypoints until it arrives at the first bin. At this point, the arm is extended and the Hand Camera is pointed down to detect either the trash bag or the bin lid. Depending on which object is detected, the state machine will follow the respective manipulation procedure from Section 2.5.2. Since removing the lid is an optional bonus challenge, it is important for the robot to recognize in real time whether this option has been chosen, as opposed to always following a strict sequence. Also, while the lid-grasping algorithm works consistently, it is still possible to fail. This design of the state machine allows the robot to detect when this happens so that another attempt at grabbing the lid may be executed. It is also possible to detect both the bag and the lid simultaneously. This exclusively occurs when the lid has already been removed but is still within the field of vision of the Hand Camera, so in this case the bag is chosen as the appropriate target object.

Once the bag has been collected, the state machine sends the robot to the collection zone through the corresponding waypoints. After placing the bag, the process is repeated for the second trash bag. At this point,

the internal timer is referenced when determining the state transitions. It is within the RoboCup@Home rules to ask for human assistance at the expense of losing some points toward the competition score [15]. If the remaining time reaches below a certain threshold, the HSR may ask for the second lid to be removed. In the context of the competition, it was more efficient to lose points for requesting human assistance than to lose points for not completing the task within the allotted time.

It is also important for the state machine to send the robot to different Nav Goals near the collection zone between the first and second drop-offs. This is to avoid dropping the second bag directly on top of the first bag. Once both bags are in the collection zone, the task is complete and the HSR announces its success.

Chapter 3

Implementation of Computer Vision for Cooking Tasks

3.1 Summary

The tasks from the RoboCup@Home competition all revolved around the concept of enabling robots to autonomously accomplish household tasks. The objective of this chapter maintains the spirit of RoboCup@Home by examining the task of cooking.

In addition to requiring precise manipulation, cooking tasks require an understanding of the cooking process itself. Cooking tasks may be considered as continuous systems, in which a set of raw ingredients becomes a ready-to-eat product through a sequence of external actions. Typically, these actions are performed by a human chef using various kitchen tools and appliances. A human chef has an understanding of how these tools and appliances function, as well as how the ingredients are affected by certain actions. This allows the chef to confirm that the cooking process is progressing as planned, or to re-evaluate the necessary actions if the ingredients reach an undesired state.

A robot chef could hypothetically avoid the need for this understanding if there was a guarantee that a given set of actions would generate the same

results every time. However, a robot operating in a dynamic household environment does not have this guarantee, so it must have a way to adapt to the current state of the ingredients in order to be reliable. For example, a reliable human chef would be able to notice if a chili pepper has exceeded its shelf life and should be replaced or omitted from the recipe. Therefore, an effective robot chef would need to be able to make similar observations and decisions.

The specific cooking task examined in this chapter is cooking a pancake. This task, like many cooking tasks, has the added complexity of being time-sensitive. If the pancake is left in the hot pan for too long, it will burn. If the pancake is removed from the heat too early, it will not be thoroughly cooked. The primary focus of this chapter is developing a method for a robot chef to evaluate the correct time to flip a cooking pancake.

It is possible to simply program an open-loop system, in which the robot chef would always flip the pancake after a constant amount of time. However, the performance of this chef would not be robust to potential variables in the cooking process. These variables, such as the consistency of the pancake mix and the heat of the pan, may result in the cooking process taking longer or shorter than the anticipated amount of time.

For the robot to react to these variables, it must have a way of sensing and interpreting its environment. One helpful feature would be the ability to acquire thermal data and determine the internal temperature of the cooking item. This feature would be applicable to essentially any task which requires heating food up to cook it. However, for a robot which is meant to be able

to perform a variety of household tasks, each additional sensor meant for a specific set of tasks increases the complexity and cost of the robot.

It would therefore be beneficial to develop a method of evaluating a cooking process using only the essential features of a domestic service robot. The platform used for this chapter is Toyota's Human Support Robot (HSR), which was also used by the Austin Villa team for the RoboCup@Home project. As seen in Figure 3.1, the HSR is equipped with a RGB-D Camera [5], which will provide the image data used to estimate the state of the pancake in this chapter.

Figure 3.1: The head of the HSR with the RGB-D Camera located above the monitor.

3.2 Contributions

This chapter discusses a project that was inspired by RoboCup@Home, but was completed separately from the competition and from the Austin Villa team. Luis Sentis, Steven Jorgensen, Nicolas Brissonneau, and the rest of the members of the Human Centered Robotics Lab at The University of Texas at Austin provided ideas and discussion points towards this project. Otherwise, this chapter discusses my contributions towards the development of a Neural Network for the use of predicting the appropriate time for a robot chef to flip a cooking pancake.

3.3 Computer Vision and Deep Learning

The concept of developing artificial intelligence to endow computers with the ability to make complex decisions has been studied for decades. Advances in the subject of machine learning has enabled computers to tackle a variety of tasks, such as classifying data, natural language processing, and making predictions about unknown data [17].

Machine learning tasks generally fall into one of three categories: reinforcement learning, unsupervised learning, and supervised learning [9]. Reinforcement learning tasks involve an agent learning to choose actions which maximize an expected reward. Unsupervised learning algorithms search for

correlations in input data without prior knowledge of expected outputs. This work primarily focuses on supervised learning, in which a set of known inputs and their corresponding labels are used to predict the labels of new inputs with unknown labels.

Deep Learning algorithms are commonly used for complex supervised learning problems. Deep Neural Networks are inspired by the functions of biological neurons which are activated in the human brain when making decisions. Similarly, artificial Neural Networks are composed of neurons which activate in response to specific inputs. For each neuron, its inputs are each multiplied by a weighting value. A bias term is then added to the sum of these weighted inputs. An activation function then operates on this value which produces the final output of the neuron. Through training on a set of data with known labels, the network is able to converge toward a set of weights and biases which minimizes the error of the network's predictions [9].

Neural Networks can be arranged in a variety of architectures which are tailored toward completing the specific task at hand. This work will focus on Convolutional Neural Networks (CNNs), which are often used in image processing tasks. Compared to basic Neural Networks, which would process each pixel individually, Convolutional Neural Networks have the added benefit of applying convolutional filters over images and thereby processing groups of pixels together [28]. These filters are able to extract topological features from the image, resulting in more reliable predictions. In the context of cooking tasks, a Convolutional Neural Network can be used to recognize features that

distinguish between a raw and cooked version of a food item.

3.3.1 Gathering Data for Training and Testing

A Neural Network must have access to a large and diverse set of relevant labelled data in order to effectively predict the label of new unlabelled data. This is analogous to the ability of humans to improve their decision-making skills for a task by gaining more experience with that particular task. The need for the training data to be varied is to allow the network to not only recognize features present in the training set, but features which are also likely to be present in new data. A Neural Network which performs well on the training set but not on testing data is likely experiencing overfitting [29].

One of the major obstacles in this work was obtaining a sufficiently large and varied dataset to train and test on. For many Deep Learning tasks, expansive databases such as ImageNet can be resourced for their vast amounts of labelled data for training and testing [13]. Unfortunately, the data needed for this project was not readily found in online databases, so the relevant images were collected manually.

The necessary data consists of images of pancake mix cooking in a pan, along with the corresponding labels indicating how far along the mix is in the cooking process. Various strategies of labeling the data is attempted throughout this project, as detailed in Sections 3.3.2 and 3.3.3. As cooking is an inherently subjective process, there is some inevitable ambiguity in the exact moment that a food item can be considered perfectly cooked. Nonetheless, all

labels were assigned within reason, such that successful predictions from the network would ensure that the final product is properly cooked.

The raw images were collected by recording videos of a pancake getting cooked and extracting the individual frames from these videos. These videos were recorded at 30 FPS, but there is often negligible change in two images taken one 30^{th} of a second apart. To reduce the number of nearly identical images and prevent the network from training on redundant data, only every 5^{th} frame was used.

It is common practice for testing data to be taken as a random subset of the available data. However, this method is not plausible for this specific task due to how the data was collected. Any randomly selected test image will have corresponding images taken of the same pancake just moments before and after the test image was taken. Test images obtained in this manner would be extremely similar to images in training data, so these test images would not be a reliable representation of how the network would perform when looking at new data. A small random subset of training images were used as validation data to evaluate the neural network after each epoch of training, but no general conclusions can be made using the validation data for the reasons stated above.

Instead, the images used for testing data were taken from entirely different videos than the images used for training data. This method accurately simulates the performance of the network in a realistic scenario, where the cooking process of a new, unique pancake would be evaluated. Chronologically

organizing the test images from a particular video allows a clear visualization of how the network performs throughout the cooking process.

3.3.2 Simple Convolutional Neural Network Classifier

For all versions of the Neural Network , the Deep Learning algorithms were developed using the **Keras** package in Python [4]. This package allows the Neural Network architecture to be quickly modified by providing simple functions to add components to the network such as convolutional layers and activation functions, as well as other tools for preprocessing images which will be referenced later in this section.

The first version of the Neural Network used in this project was adapted from an online tutorial on using **Keras** to generate Convolutional Neural Networks [26]. The Convolutional Neural Network in this tutorial was made for the MNIST Fashion dataset, which contains 70,000 28x28 grayscale images of clothing items in one of ten categories [31]. For the purpose of this project, the input images of cooking pancakes were resized to 64x64 and kept as RGB images. The input layer was therefore modified to accept an input image with dimensions (64,64,3).

For this preliminary version of the Convolutional Neural Network, the input images were labelled as 0 if the pancake was not sufficiently cooked, and labelled as 1 if the pancake was ready to be flipped to cook the other side. The images were divided into 'Training', 'Validation', and 'Testing' directories, and then subdivided into 'Uncooked' and 'ReadyToFlip' directories based on

the corresponding label. The raw images extracted from the video recordings were resized to 64x64 and had their pixel values normalized between 0 and 1 before being used as inputs to the network. An example of one input is shown in Figure 3.2. The images were purposely collected with the stove-top and background items visible, as it was intended to simulate what the robot may see in the background while cooking. This feature is addressed further in Section 3.3.3.1.

Figure 3.2: A raw image which was resized and used as an input to the neural network for training. This image was labelled as 0, indicating the pancake has not yet been sufficiently cooked.

3.3.2.1 Architecture

This network consists of three convolutional layers, which in sequential order contain 32, 64, and 128 filters. Each filter has a kernel size of 3x3 and a stride of 1. As described previously, these convolutional layers are primarily what separates a Convolutional Neural Network from a more generic Neural Network, as they evaluate clusters of pixels as a group rather than individually.

These convolutional layers are each followed by an activation function. One of the most commonly used activation functions in modern Neural Networks is the Rectified Linear Unit (ReLU). This activation function can be represented as $f(x) = max(0, x)$, in which the output is the same as the input x unless x is negative, in which case the output is 0. The use of nonlinear activation functions enables the network to make complex decisions which could not be accomplished with a linear mapping. ReLU has been shown to converge more quickly than other nonlinear activation functions such as the Sigmoid and Hyperbolic Tangent functions [13].

One downfall of ReLU is that it results in a gradient of 0 when operating on a negative input. Gradients are used by Deep Learning algorithms to systematically modify the weights and biases to optimize the network. A gradient which is permanently zero can therefore interfere with the learning process. One method of approaching this issue is using the Leaky ReLU function. Instead of giving an output of 0 for a negative input, a negative input to a Leaky ReLU function is multiplied by a small value α [14]. This allows for a small gradient to remain present at all times. Due to this benefit, the

Leaky ReLU activation function with an α value of 0.1 is chosen to be used after each convolutional layer.

Following each convolutional layer and Leaky ReLU activation function is a Max Pooling layer. This layer uses a kernel size of 2x2 as well as a stride of 2. This Max Pooling function divides each slice of its input into 2x2 regions and outputs the maximum value in this region. This layer reduces the dimensions of the input, thereby requiring less parameters to be tuned in succeeding layers. For example, the first convolutional layer has an output of size (64,64,32), corresponding to the size of the original 64x64 image and the 32 convolutional filters. The Leaky ReLU layer does not change any dimensions, but the Max Pooling layer has an output of reduced size (32,32,32) The use of the max function keeps the features associated with the most highly activated neurons. This has the added benefit of preventing the model from overfitting to less relevant features which are specific to the training data [21].

To further reduce overfitting, a Dropout layer is added after each Max Pooling layer which randomly selects a fraction p of neurons to temporarily disconnect from the network. While it may be counter-intuitive to disconnect components of a model, it provides the benefit of reducing dependency on specific features [29]. A network should be able to recognize an image even if it does not have all of its expected features. Overfitting is often a consequence of the network learning features of the training set which are not generally applicable to new data. The dropout layers force the network to continue learning without being dependent on all of the learned features. A p value of

0.5 is generally effective for these Dropout layers [29], so 0.5 was the p value chosen for this network.

After the last Dropout layer, a Flatten layer is added which organizes the 3-dimensional output into a 1-dimensional array of neurons. For this network, the final output of the last Max Pooling Layer is (8,8,128), which gets flattened to a 1-dimensional array of size 8192. This flattened layer is then connected to a Dense layer of size 128. This is the type of layer which is typically found in a generic Neural Network, where each neuron is connected to every neuron in the previous layer. This Dense layer is followed by another Leaky ReLU layer and Dropout layer.

The output layer is another Dense layer, with one output which corresponds to the prediction of the network. Since the data labels are either 0 or 1, the output of the network should be between 0 and 1. For this reason, the Sigmoid activation function was used as it restricts the output to this desired range. An output close to 0 indicates that the network believes the pancake is not sufficiently cooked, while an output close to 1 indicates that the network is confident that the pancake is ready to be flipped.

3.3.2.2 Preliminary Results

Overall, the first attempt at using a Convolutional Neural Network to predict the state of a cooking pancake had poor results, though there were some promising trends indicating that this methodology still had potential. To evaluate the model, images extracted from videos which were not used for

the training dataset were tested. The results of one of these test cases are shown in Figure 3.3. In this trial, the pancake was ready to be flipped after 52 seconds.

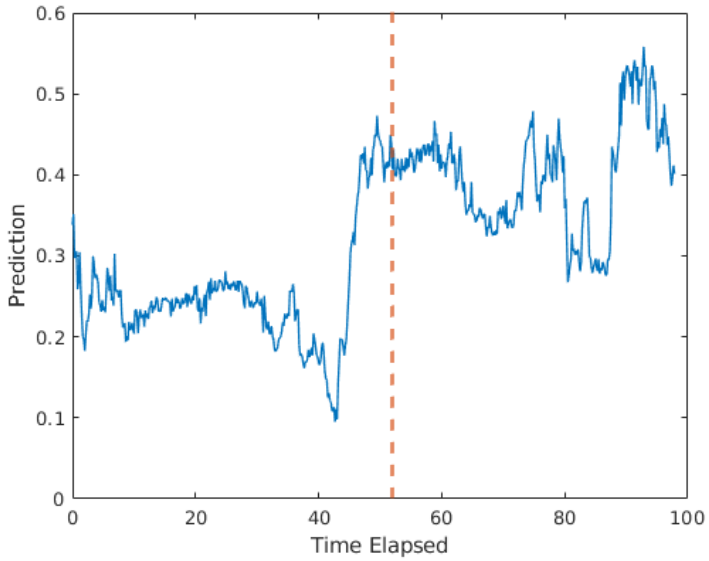

Figure 3.3: The results from the first attempt at using a Convolutional Neural Network to predict the state of a cooking pancake. The dashed orange line indicates the actual moment that the pancake was ready to be flipped.

It is evident that the predictions of this model do not accurately match the real system. For approximately 40 seconds at the beginning of the trial, there is a slight downward trend in the prediction. This is an undesired result, since the model would ideally begin at a prediction near 0 when the pancake mix is first poured onto the pan, and the prediction would be expected to gradually increase towards 1 as the pancake cooks.

One promising feature of these results is that there is a notable rise in the prediction near the time where the pancake becomes ready to be flipped. This indicates that the network does in fact recognize some features that distinguish a sufficiently cooked pancake from a partially cooked pancake. However, this prediction should approach a value of 1, but instead it never reaches a value much greater than 0.5.

One plausible explanation for this behavior of the model has to do with the fact that the appearance of the pancake does not have a particularly noticeable change after the point that it is ready to be flipped. When the pancake mix is first poured onto the hot pan, bubbles begin to form and pop on the surface as the mix goes through the cooking process. After some time, the mix starts to solidify and new bubbles stop forming. This occurs around the point in time that the pancake is ready to flip, meaning that while the bottom of the pancake continues to cook after this point, the visible surface no longer experiences significant changes in appearance. This behavior causes a pancake that is labelled as 1 to look very similar to a pancake moments away from being ready, which would be labelled as 0. This likely makes it difficult for the network to distinguish between these two cases of pancake images.

Another undesirable feature of the results in Figure 3.3 is that the predictions do not follow a clear and stable trend. Instead, there are sporadic spikes in the prediction throughout the trial. While a very similar Neural Network model was able to perform with great accuracy on the MNIST Fashion dataset [26], this model requires further adaptations for the specific task in

this project. The issues of these preliminary results are addressed throughout Section 3.3.3.

3.3.3 Intermediate Versions of Deep Learning Algorithms

This section will look at the evolution of the Neural Network used in this project. Building off of the simple Convolutional Neural Network from Section 3.3.2 which had minimal effectiveness for this task, each version of the Neural Network preceding the final version will be evaluated and analyzed to determine how the performance was improved in later versions.

3.3.3.1 Implementing Bounding Boxes

One clear issue of the preliminary model was its inability to perform on new datasets. A likely cause of this is overfitting to features the training dataset. The preliminary model was trained on image inputs similar to that shown in Figure 3.2, which includes some of the stove-top and other background objects. Inputting these images directly into the network results in the network potentially learning features of the background which are irrelevant to the state of the pancake.

The ability to programmatically generate bounding boxes around the pancake would allow the background to be easily removed from training images, as well as allow the robot to ignore the background while cooking. For this reason, a Python script was written which converts a raw image into an image of the pancake without the background. This script was written largely

with the help of the **OpenCV** Python package, which provides helpful tools for Computer Vision tasks [3].

Since the bounding-box-generating algorithm is intended to work in real time while the robot is cooking, the first step of this algorithm was to scale down the image to allow quicker computation. A scaling factor of 0.25 for the image height and width was found to significantly speed up the algorithm while maintaining precision of the resulting bounding boxes.

Next, **OpenCV** was used to apply K-Means Clustering to the scaled-down image. K-Means Clustering is a form of unsupervised learning [17], in which data is split into k clusters, each centered about a mean value. The goal of the K-Means algorithm is to determine the mean values, or centroids, which minimize the error between the centroids and the elements of the respective clusters [8]. For all of the images collected, a k value of 2 was sufficient in this algorithm. The original image from Figure 3.2 after K-Means Clustering is shown in Figure 3.4.

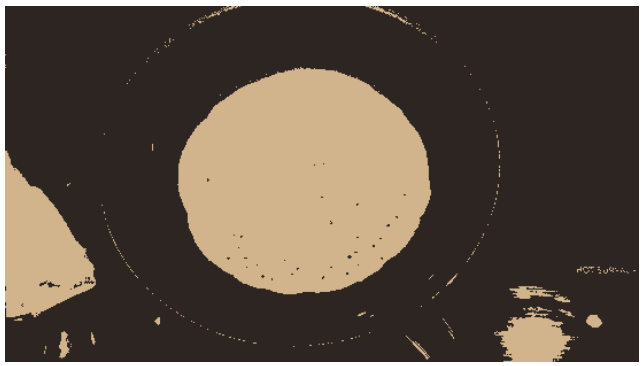

Figure 3.4: The output of applying K-Means Clustering with a k value of 2 to a scaled-down version of the image in Figure 3.2.

Since it is possible for objects in the background to have a similar color as the pancake, K-Means Clustering cannot reliably separate the pancake into its own cluster. However, this algorithm does reliably distinguish between the pancake and the pan. Through **OpenCV**, it is possible to find the contours in a binary image. To make the image binary, it is converted to grayscale and a threshold is applied to divide the image into pixels above and below this threshold value. Since the pan is dark and the pancake is relatively light, the cluster containing the pan will reliably be on the opposite side of the threshold compared to the cluster containing the pancake.

Now that the image has been made binary, the contours of the pancake can be found. Since other regions have pixels below the threshold, extraneous contours are found which do not belong to the pancake. The first step in identifying the correct contour is to filter out all contours below a certain size, as it is known that the contour of the pancake should be relatively large. It is also known that the pancake should be approximately at the center of the image, so from the remaining contours, the contour with the pixels most clustered around the center is predicted to be the contour associated with the pancake.

With the correct contour identified, **OpenCV** can form a bounding box around this contour. Finally, the coordinates of this bounding box are scaled up according to the scaling factor used in the first step of this algorithm and

then applied to the original unscaled image. This algorithm was found to consistently produce accurate bounding boxes. Figure 3.5 shows the detected contours as well as the contour predicted to correspond to the pancake. Figure 3.6 shows the final output of this algorithm.

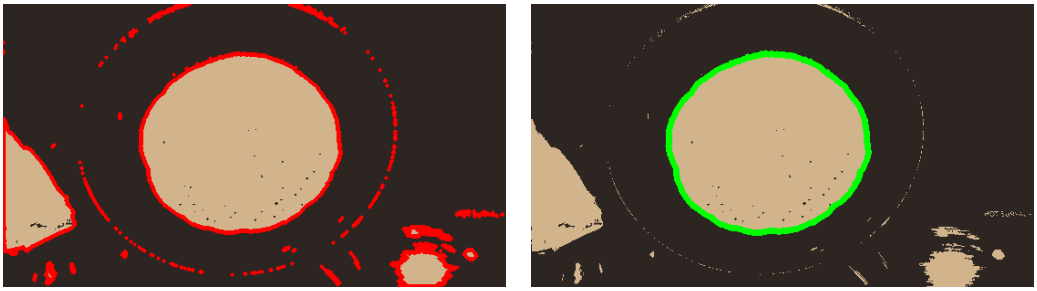

Figure 3.5: On the left, all detected contours are shown in red. On the right, the contour which was predicted to be associated with the pancake is in green.

Figure 3.6: The output of applying the bounding-box-generating algorithm to the image in Figure 3.2.

3.3.3.2 Implementing Data Augmentation

Another way to avoid overfitting is to increase the size and variety of the training dataset. Due to the difficult nature of obtaining large amounts of effective training data for this project, Data Augmentation techniques were used to approximate the effect of gathering more data. Data Augmentation refers to methods of modifying the available data to generate unique training data [27]. The **Keras** package allows some of these methods to be easily applied to the training images, such as rotating, flipping, changing brightness levels, and shifting the image height and width. After applying bounding boxes and Data Augmentation to the training data, the model was retrained.

Compared to the preliminary results, the updated results shown in Figure 3.7 show a much clearer positive trend as the pancake cooks. The prediction steadily increases over time, and the spikes in the prediction are more subdued. Also unlike the preliminary results, the prediction here begins at approximately 0, which is an appropriate prediction to make at the start of the cooking process. These improvements clearly show the benefits of implementing bounding boxes and Data Augmentation. However, there is still the issue of the prediction struggling to exceed 0.5 in this model.

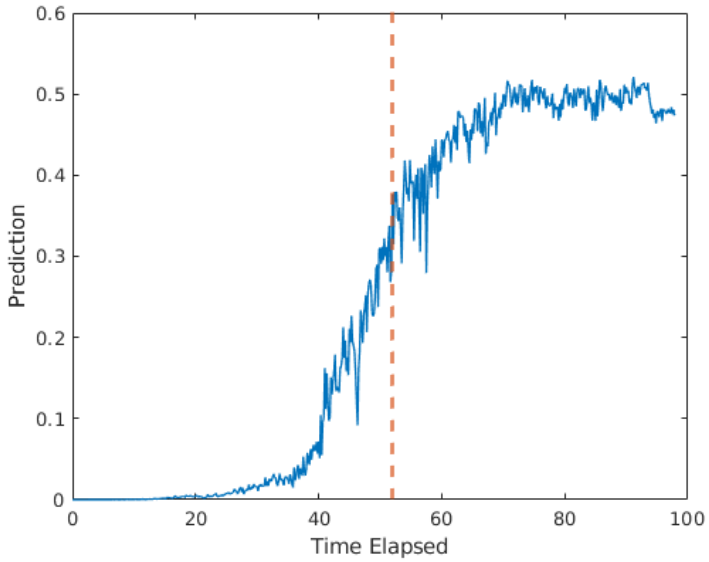

Figure 3.7: The results from the using a Convolutional Neural Network to predict the state of a cooking pancake after applying bounding boxes and Data Augmentation to the images. The dashed orange line indicates the actual moment that the pancake was ready to be flipped.

3.3.3.3 Remodelling as a Regression Problem

To begin resolving the issue of predictions not approaching 1, it is first noted that details of the cooking process are lost when treating it as a binary process. Using the current methodology, an image of pancake mix which has just been poured on the pan ends up with the same label as an image of a pancake which is just seconds away from being ready, even though these images have clear and relevant differences.

Cooking is a continuous process, and it is therefore reasonable to model

it as such for this project. In this way, the cooking process is similar to the aging process, which may be split into broad classifications such as Young and Old, but overall is a continuous process. Convolutional Neural Networks have been used for age estimation using a regression model, as this model recognizes the ordinal relationships between states [18]. From now on, this cooking problem will be modelled as a regression problem.

To generate labels for the regression problem, the moment in time where the pancake mix is poured onto the pan is given a label of 0 and the moment where the pancake becomes ready to flip is given a label of 1. From there, the intermediate values are linearly interpolated. For example, in a trial which takes 60 seconds for the pancake to cook, the image of the pancake after 45 seconds of cooking would receive a label of 0.75, where as it would previously have a label of 0. Similarly, an image of this pancake after 75 seconds would receive a label of 1.25. This label can be thought of as how much the pancake has been cooked. Since it is now possible for the label to exceed a value of 1, the Sigmoid activation function at the output layer is replaced by a linear activation function which does not place bounds on the prediction of the model.

Since there are now an infinite number of possible labels, it is no longer possible to divide images into directories based on their label. Instead, a **Pandas** dataframe was used to organize the data [16]. The dataframe contained one column of image filenames, and another column containing the corresponding label of each image. The **Keras** package allows Data Augmentation to be easily applied to data presented through a **Pandas** dataframe for use in Con-

volutional Neural Networks. The results of applying this model to a test case are shown in Figure 3.8.

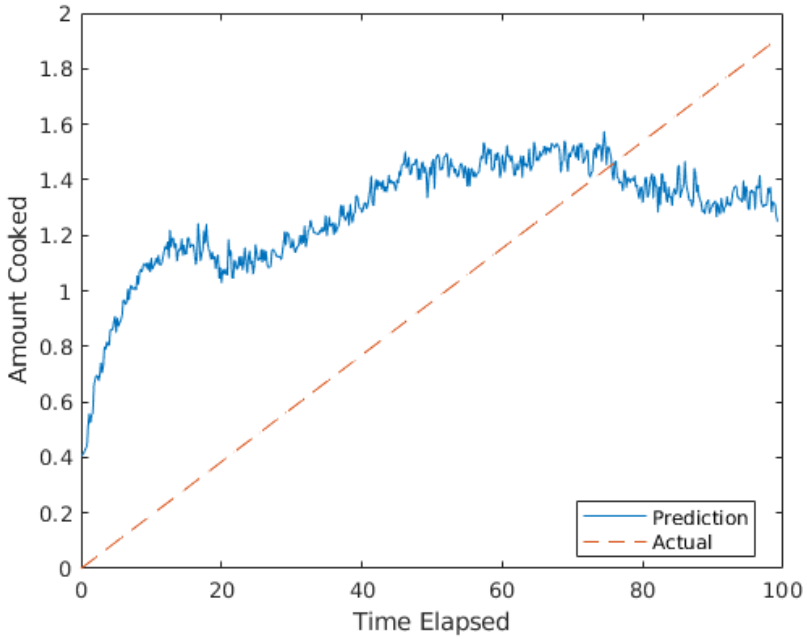

Figure 3.8: The results from the using a Convolutional Neural Network to predict the state of a cooking pancake after remodelling the problem as a regression. The solid blue line represents the predicted amount cooked. The dashed orange line indicates the labelled amount cooked.

While this change did effectively remove the apparent boundary at a prediction of 0.5, there is still a highly apparent inaccuracy in the model when compared to the labelled data. The predictions of this model begin at around 0.4 instead of the preferred value of 0. Additionally, the notable rise in the

predictions at around 40 seconds which can be seen in Figures 3.3 and 3.7 is no longer visible. This is an undesirable outcome, as this rise seemed to indicate that the previous models had an understanding of when the pancake was approaching completion. Overall, it is clear that more steps need to be taken to improve the model.

3.3.3.4 Adding Convolutional Layers and Batch Normalization

The original images extracted from the video recordings has a size of 1080x1920, so the resizing to 64x64 images caused a significant loss in resolution, even for the images with bounding boxes applied. Increasing the size of the input image would require more parameters to be trained in the network, but it is plausible that the higher resolution would improve performance.

Each convolutional layer is followed by a Max Pooling layer which reduces the dimensions of the input to that layer. This essentially places a limit on how many convolutional layers can be put in a model before the dimensions become too small. Increasing the size of the input image would thereby allow more convolutional layers to exist in the network. This would again increase the complexity of the model, but would allow more refined features to be extracted from the images which has the potential to improve performance. The size of the input image, the number of convolutional layers, and the size of these convolutional layers are hyperparameters which are assigned through trial and error to determine which values provide the best results.

It was also found that standard Dropout layers do not generally re-

duce overfitting when used between convolution layers as it does for Dense layers [30]. For this reason, the Dropout layers which immediately followed convolutional layers were removed from the model. In their place, Batch Normalization layers were added which speed up learning by normalizing batches of data as they pass through the hidden layers of the network [10]. This layer mimics the common practice of normalizing the input to the network before processing it.

3.3.3.5 Adding Elapsed Time as an Additional Input

Up to this point, the Neural Network has accepted only one input, which is an RGB image of a pancake as it is being cooked. Past work has shown that Neural Networks can benefit from having multiple inputs which may be of mixed data types [1]. With this in mind, new relevant inputs are considered to be added alongside the image data.

One highly relevant input would be temperature data of the pancake. However, as mentioned in Section 3.1, this data may not be readily obtainable for a general-purpose domestic service robot due to hardware limitations. This project was designed around the use of Toyota's Human Support Robot, through which thermal sensing is not available [5].

Another relevant input is the amount of elapsed time, which can be measured without the use of additional hardware. A human chef typically has an understanding of the amount of time generally required for a food item to cook, and therefore considers how much time has currently passed when

deciding whether that item is sufficiently cooked. It is therefore reasonable to include this measure as an input to the neural network.

To accomplish this, a separate branch is made which runs in parallel to the original Convolutional Neural Network. This branch accepts elapsed time in seconds as an input, which is normalized by dividing by 150. This normalizing factor was chosen because none of the cooking trials involved a pancake being cooked for over 150 seconds, thereby bounding this input between 0 and 1.

The outputs of both branches were concatenated and then connected to a new Dense layer where the time data was processed in conjunction with the processed image data. This Dense layer is followed by a ReLU activation function and then connected to another Dense layer with size 1, which acts as the final output layer of the new model. Since the output of the Convolutional Neural Network branch is no longer the output of the overall model, the activation function of the final Dense layer in that branch is switched from linear to ReLU. Instead, the linear activation function is applied to the last Dense layer after the branches are concatenated.

As mentioned in Section 3.3.3.4, the input image size and number of convolutional layers were modified to improve results. The hyperparameters which were found to produce the best results were an image size of 224x244 and 4 convolutional layers of size 32, 64, 128, and 256, sequentially.

This modification of the Neural Network showed a significant and im-

mediate increase in the accuracy of the model. With the added information of how much time is taken for the pancakes to cook, the predicted cooking trends are much more similar to the actual trends than in previous versions of the network. One visible flaw of this model is that the network appears to find a strong correlation between the prediction and elapsed time, while the predictions are largely invariant to the image data. The results of running the model on three test cases are shown in Figure 3.9.

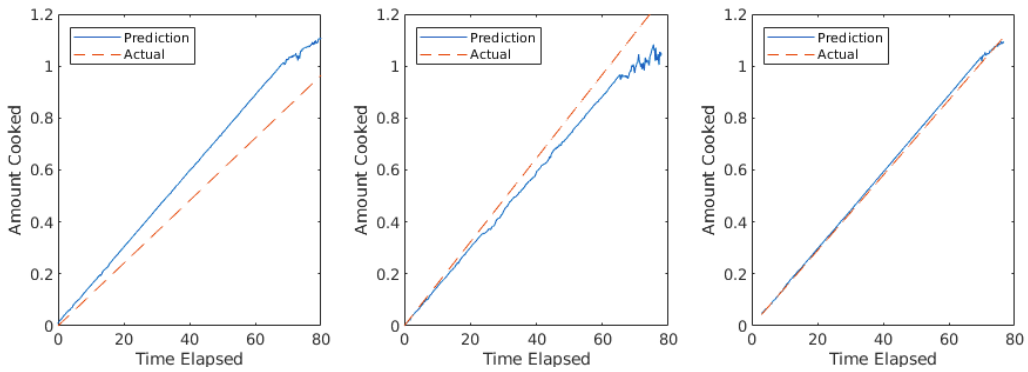

Figure 3.9: The results from the using a Convolutional Neural Network to predict the state of a cooking pancake after adding elapsed time as an input to the model. The solid blue line represents the predicted amount cooked. The dashed orange line indicates the labelled amount cooked. The labels of the rightmost plot do not pass through the origin because the pancake had been cooking for 3 seconds before the video recording started.

In the rightmost case from Figure 3.9, the prediction is highly accurate to the real data. However, in the leftmost case the model overestimates the speed at which the pancake cooks, and this speed is underestimated in the

middle case. In each case, the model predicts that the pancake is ready to be flipped after 65 to 70 seconds, but the actual trials took 83, 62, and 66 seconds to cook. This version of the model still struggles to accurately predict the behavior of pancakes which do not take the expected amount of time to cook.

3.3.3.6 Revising Error Metric in Regression Model

The final major change to the model was to redefine the label as the time left to cook, rather than the amount cooked. This change was made primarily because it is irrelevant for this task to know what fraction of the cooking process has been completed while the pancake is partially cooked. All the robot chef needs to know is the moment that the pancake should be flipped, so the error metric was redefined to reflect this need. In these new versions of the Neural Network, the model will predict how much time is left before the pancake should be flipped. When this prediction reaches zero, the robot chef will know to flip the pancake. Additionally, the metric is not as arbitrary as before, because the labels for amount cooked assumed the cooking process to be linear.

The elapsed time labels were again divided by 150 to normalize the values between 0 and 1. Additionally, a Dropout layer was added after the concatenated outputs of the Convolutional Neural Network and time branches, with the intent of reducing the dependency of the model on the time data which was found in Figure 3.9. The results of these changes are discussed in Section

3.3.4, where the best-performing version of the Neural Network is described.

3.3.4 Final Neural Network

Of all the models tested for this project, the most effective one accepted an input of a 224x224 RGB image which was processed through a Convolutional Neural Network branch, as well as an input of elapsed time which was processed through a separate branch. The architecture of the image processing branch is shown in Figure 3.10, while the remainder of the model is shown in Figure 3.11.

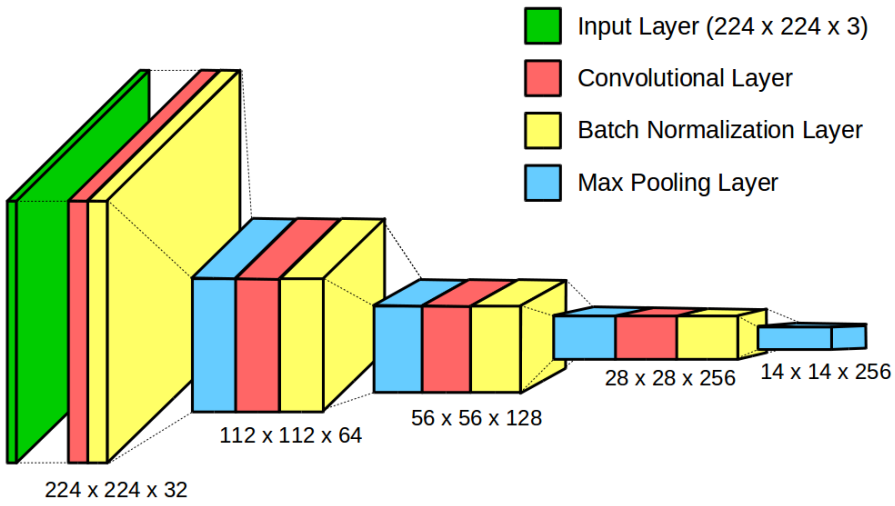

Figure 3.10: Architecture of the image processing branch of the Neural Network. The output of the last Max Pooling layer is flattened and used as the 'Processed Image' input in Figure 3.11

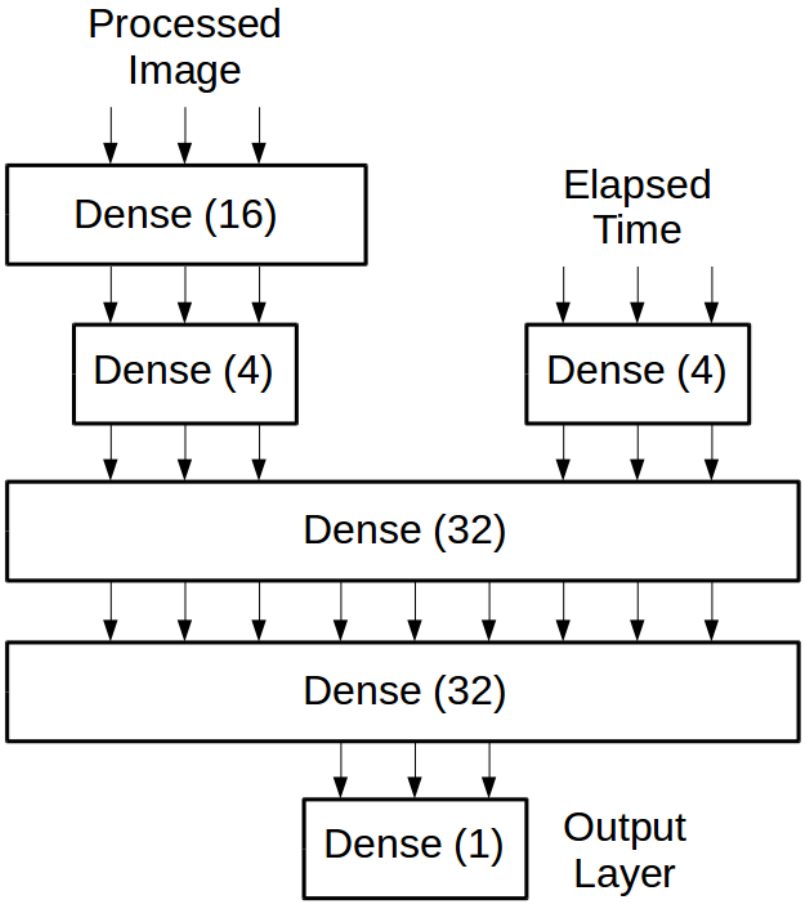

Figure 3.11: A representation of the concatenation of the two input branches leading up the the final output of the network.

The Convolutional Neural Network branch contained 4 convolutional layers of sizes 32, 64, 128, and 256, sequentially. Each convolutional layer used a kernel size of 3x3 and was followed by a ReLU activation function, then a

Batch Normalization Layer, and then a Max Pooling layer with a kernel size of 2x2 and stride of 2. After the last Max Pooling layer, the neurons were flattened and connected to a Dense layer of size 16 with a ReLU activation function. This layer is followed by another Batch Normalization layer and a Dropout layer before connecting to the output layer of the Convolutional Neural Network branch. The output layer is a Dense layer of size 4 with a ReLU activation function.

Running in parallel to the Convolutional Neural Network branch, the time branch simply connects the time input to a Dense layer of size 4. This Dense layer is used to match the output of the Convolutional Neural Network branch, with the intent of balancing the contribution of each of these branches.

The outputs of the two branches are concatenated and then connected to a Dense layer of size 32 with a ReLU activation function. A Dropout layer is inserted before a second Dense layer with size 32 and a ReLU activation function. Finally, the output layer of the model is a Dense layer with size 1 and a linear activation function, which outputs the normalized prediction of how much time is left for the pancake to cook. This output is then multiplied by the normalizing factor of 150 to obtain the true predicted value. The results of this model for three test cases are shown in Figure 3.12.

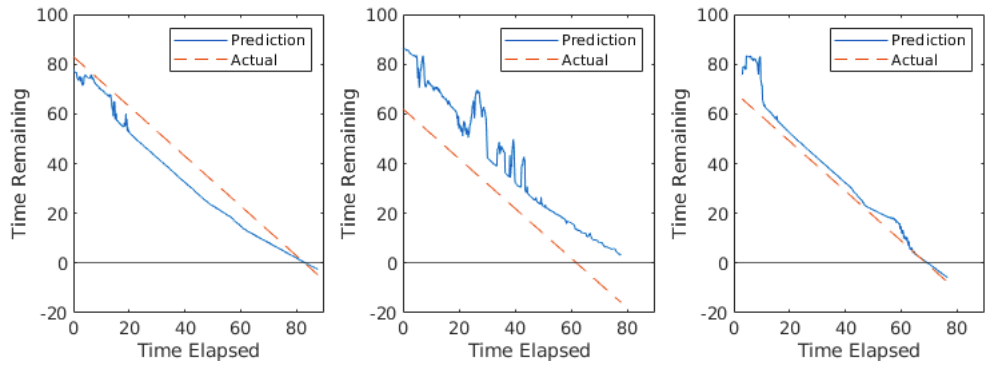

Figure 3.12: The results from the final version of a Convolutional Neural Network used to predict the state of a cooking pancake. The solid blue line represents the predicted time left to cook. The dashed orange line indicates the labelled time left to cook.

This model appears to initialize with a guess of approximately 80 seconds in each case, whether or not this is an accurate prediction. This is likely due to the fact that not enough of the cooking process has yet taken place for the model to accurately predict the rate at which the pancake is being cooked. However, this behavior does not directly affect the viability of this model. This is because the robot will make the decision to flip the pancake only when the prediction reaches zero, so the early predictions are essentially irrelevant to the decision-making process.

One of the test cases included a pancake made from a mix with a noticeably less dense consistency than average. The results from this test case can be seen in the center plot of Figure 3.12. This case was intentionally chosen to evaluate how the model would perform in unusual cases, and the results from

the plot indicate that this performance is not ideal. This is likely due to a lack of large amounts of available data which would expose the network to more atypical cases during the training phase.

In the leftmost and rightmost cases in Figure 3.12, the lines representing the actual and predicted labels cross the x-axis at virtually the same time, which is exactly the desired outcome of this model. In the leftmost case, the model initially underestimates the time required to cook the pancake, but gradually converges to the actual time remaining. Similarly, in the rightmost case, the model initially expects the pancake to take nearly 80 seconds to cook, but quickly adapts and recognizes that the pancake was cooked in just 66 seconds. The ability of this model to adapt to the input images, rather than making very similar predictions in every trial as in Section 3.3.3.5, shows that the implementation of Computer Vision for this task is more effective than simply flipping the pancake after a fixed amount of time.

Chapter 4

Conclusion and Future Work

4.1 RoboCup@Home

The RoboCup@Home competition promoted finding reliable and quickly implementable solutions to a variety of problems. While the skills for that project were developed as an answer to the specific collection of tasks provided by the RoboCup Federation, the methodologies used may also be implemented for more generalized applications.

With additional focus on specific tasks, it is possible to further enhance performance and develop useful algorithms. One way performance may be improved upon is by making the actions of the robot more time efficient. For example, in the current system the HSR comes to a complete stop at each waypoint. A more advanced Navigation stack, such as that from the STRANDS Project[1] may be implemented to generate a more optimal trajectory. Also, the current trial-and-error lid grasping method may be replaced by an algorithm to systematically detect the handle orientation through Computer Vision, such that the handle may be grabbed correctly on the first attempt.

[1] `https://github.com/strands-project/strands_navigation`

4.2 Improving The Neural Network

The performance of the Neural Network in Chapter 3 was hindered by the lack of access to large amounts of training and testing data. Even with the use of Data Augmentation, it is plausible that the network would require thousands of unique trials to learn from in order to become adequately robust in a dynamic real-world setting. This amount of data was not feasible to obtain for this project, but future research on this model with ample training data could show even more promising results.

The extra data could also be used for more thoroughly testing the model. Currently, the model is evaluated by looking at individual trials, as there were not enough available test trials to perform meaningful statistical analysis over a set of trials.

One of the features of a cooked pancake is the lack of new bubbles being formed at the surface. When looking at only one image of a pancake, it is impossible to tell how the surface texture has changed relative to previous images. It may therefore be possible to improve the model by evaluating a sequence of images instead of only one image at a time. For this purpose, the use of a Recurrent Neural Network may be applicable for cooking tasks, as they are generally used to model sequenced data by using previous outputs as inputs [2].

In this project, the labels were assigned based on the amount of time remaining for the pancake to be cooked, as this labelling provided the best re-

sults. However, other methods of labelling may be beneficial in other contexts. For example, a model which could reliably classify a food item as being sufficiently cooked would effectively output a set of probabilities corresponding to the predicted state of the food item. These probabilities could be implemented as part of a Partially Observable Markov Decision Process (POMDP), or Belief Space Planning algorithm. POMDPs are a form of reinforcement learning which attempts to estimate states and make decisions which maximize an expected reward [25]. Flipping a pancake in a 'Cooked' state is an example of an action which would provide a positive reward in this context, while letting the pancake reach a 'Burnt' state would result in a negative reward.

Similarly, the 'Amount Cooked' metric may be applicable in contexts where the optimal amount to cook the food item is largely decided by personal preference. For example, when cooking burgers, a label of 0.7 may be applied to a medium-rare patty, while a label of 1 would be applied to a well-done patty. This would allow the robot to readily adjust to the preference of a specific person.

4.3 Manipulation

The project from Chapter 3 focused on enabling a robot to understand the cooking process through Computer Vision and Deep Learning. A logical next step would be to implement these learning algorithms on a robot with the manipulation capacity to perform the actual cooking task.

This project was made with Toyota's Human Support Robot in mind, as

this robot has the relevant features which enable it to manipulate items in the kitchen and around the house. Depending on the availability of equipment, the manipulation tasks may be tested in either a real or a simulated environment. The required tasks would include scooping up the pancake with a spatula, flipping the pancake to cook both sides, and placing the cooked pancake on a plate, ready to be served. Figure 4.1 shows an example of the robot operating in a simulated environment through the Gazebo software.

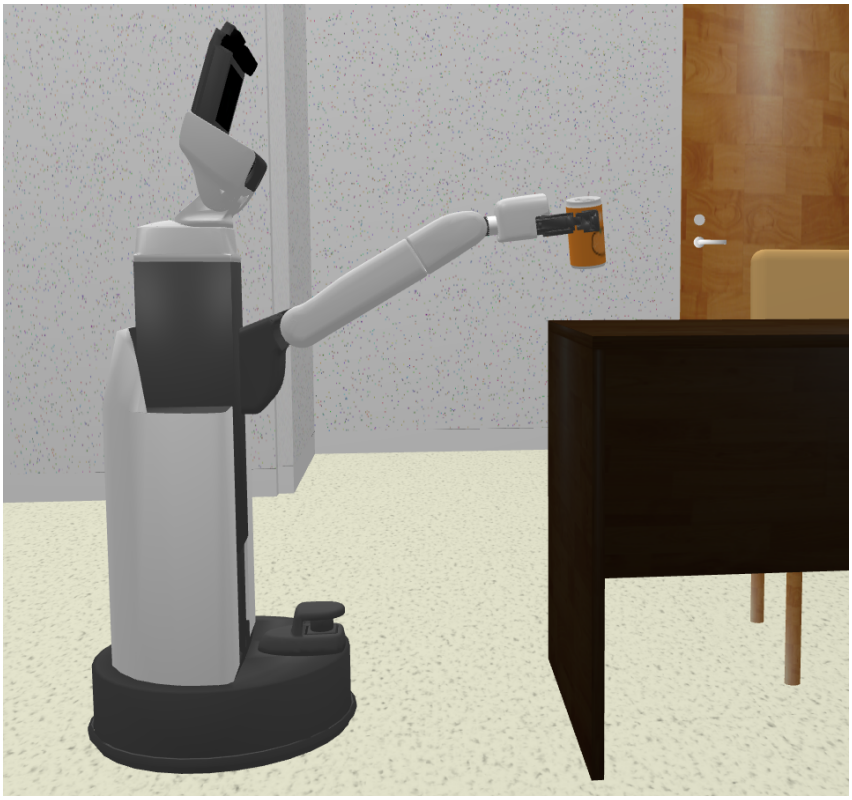

Figure 4.1: The Toyota Human Support Robot grabbing an object in a simulated environment.

These manipulation tasks would also require additional Computer Vision components. One example is the robot's need to be capable of determining the position of the pancake in space. While the current methodology includes an algorithm to generate a bounding box around the pancake, this only provides information in 2 dimensions. It is possible to combine these 2D bounding boxes with point cloud data from the RGB-D Camera to approximate 3D bounding boxes. This is methodology that was applied with YOLO Object Detection software [19] in the RoboCup@Home project. Similarly, the robot must be able to determine the locations of the pan and plate where the pancakes is to be placed.

4.4 Closing Remarks

Overall, the work done for RoboCup@Home demonstrates the progress and capabilities of robotic systems in realistic domestic environments. It can be expected that future competitions will continue to address natural Human-Robot Interaction and avenues for robots to improve quality of life. Expectations will continue to rise as the limits of domestic service robots are pushed and their applications in the real world are further recognized.

Similarly, the results in Chapter 3 have indicated that Computer Vision and similar Machine Learning strategies can be applied to enable domestic service robots, such as Toyota's Human Support Robot, to improve performance in cooking tasks. While the current model does not robustly perform as well as a human can perform, it is shown that Machine Learning strategies may be

used to give robots an understanding of cooking processes. This project opens up avenues to further explore enabling robots to accomplish similar tasks. The findings in this project are not only applicable to cooking pancakes, but can be readily expanded for use in a variety of cooking applications.

Bibliography

[1] Eman H. Ahmed and Mohamed N. Moustafa. House price estimation from visual and textual features. *Proceedings of the 8th International Joint Conference on Computational Intelligence*, September 2016.

[2] Afshine Amidi and Shervine Amidi. Cs 230 - deep learning: Recurrent neural networks cheatsheet.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] François Chollet et al. Keras. `https://keras.io`, 2015.

[5] Toyota Motor Corporation. Toyota shifts home helper robot R&D into high gear with new developer community and upgraded prototype, July 2015.

[6] Toyota Motor Corporation. HSRB development manual, 2015-2019. `https://docs.hsr.io/hsr_develop_manual_en/`.

[7] Toyota Motor Corporation. HSRB user manual, 2015-2019. `https://docs.hsr.io/hsrb_user_manual_en/`.

[8] Imad Dabbura. K-means clustering: Algorithm, applications, evaluation methods, and drawbacks. *Towards Data Science*, September 2018.

[9] Ke-Lin Du and M. N. S. Swamy. *Neural Networks and Statistical Learning*. Springer, London, 2nd edition, 2019.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Computing Research Repository*, March 2015.

[11] Yuqian Jiang et al. LAAIR: A layered architecture for autonomous interactive robots. *In Proceedings of the AAAI Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy*, November 2018.

[12] Stefan Kohlbrecher, Johannes Meyer, Oskar von Stryk, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. *IEEE International Symposium on Safety, Security and Rescue Robotics*, November 2011.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, May 2017.

[14] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *International Conference on Machine Learning*, 2013.

[15] Mauricio Matamoros, Caleb Rascon, Sven Wachsmuth, Alexander William Moriarty, Johannes Kummert, Justin Hart, Sammy Pfeiffer, Matthijs van

der Brugh, and Maxime St-Pierre. RoboCup@Home 2019: Rules and regulations. http://www.robocupathome.org/rules/2019_rulebook.pdf, 2019.

[16] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[17] Afshin Rostamizadeh Mehryar Mohri and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2nd edition, 2018.

[18] Zhenxing Niu, Mo Zhou, Le Wang, Xinbo Gao, and Gang Hua. Ordinal regression with multiple output cnn for age estimation. *IEEE Conference on Computer Vision and Pattern Recognition*, June 2016.

[19] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Computing Research Repository*, May 2016.

[20] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *Computing Research Repository*, December 2016.

[21] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 1999.

[22] Open Robotics. About ROS. https://www.ros.org/about-ros/.

[23] Ioan Sucan Sachin Chitta and Steve Cousins. MoveIt! *IEEE Robotics & Automation Magazine*, March 2012.

[24] Rishi Shah et al. Solving service robot tasks: UT Austin Villa@Home 2019 team report. *AAAI Fall Symposium on Artificial Intelligence and Human-Robot Interaction for Service Robots in Human Environments*, November 2019.

[25] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 2013.

[26] Aditya Sharma. Convolutional neural networks in Python with Keras. *DataCamp Community*, December 2017.

[27] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, July 2019.

[28] Patrice Y Simard, Dave Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. *Seventh International Conference on Document Analysis and Recognition*, August 2003.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, June 2014.

[30] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. *IEEE Conference on Computer Vision and Pattern Recognition*, June 2015.

[31] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *Computing Research Repository*, 2017.